Pipelined Reconfigurable Accelerator for Ordinal Pattern Encoding

Ce Guo\textsuperscript{1} \hspace{0.5cm} Wayne Luk\textsuperscript{1} \hspace{0.5cm} Stephen Weston\textsuperscript{2}

\textsuperscript{1}Imperial College London

\textsuperscript{2}Maxeler Technologies

ASAP 2014
Next value: A, B or C?
Next value: A, B or C?
Next value: A, B or C?
Observation: not all time series are predictable due to randomness

Problem: different actions required for different predictability

Solution: measure predictability using ordinal analysis
Overview

- Two-level ordinal pattern encoding scheme with pipeline-friendly properties
  - Minimised data transmission
  - Simple data operations
  - Free from post-processing

- Pipelined accelerator for ordinal pattern encoding

- Experimental evaluation:
  - Up to 12 times faster than four-core CPU
  - Up to 4 times more energy-efficient than four-core CPU
Ordinal Analysis

- Ordinal analysis: discover useful information from the distribution of ordinal patterns
- Ordinal pattern by example

<table>
<thead>
<tr>
<th>Time series</th>
<th>Ordinal pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 2 3 8 1 4</td>
<td></td>
</tr>
<tr>
<td>6 2 3 8 1 4</td>
<td></td>
</tr>
<tr>
<td>6 2 3 8 1 4</td>
<td></td>
</tr>
<tr>
<td>6 2 3 8 1 4</td>
<td></td>
</tr>
<tr>
<td>6 2 3 8 1 4</td>
<td></td>
</tr>
</tbody>
</table>
# Ordinal Analysis

## Histogram and distribution:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Count</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Pattern 1" /></td>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td><img src="image2" alt="Pattern 2" /></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><img src="image3" alt="Pattern 3" /></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><img src="image4" alt="Pattern 4" /></td>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td><img src="image5" alt="Pattern 5" /></td>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td><img src="image6" alt="Pattern 6" /></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
The distribution of ordinal patterns reflects chaotic properties of the time series.

Application of this distribution:
- Detection of dynamic changes
- Permutation entropy
Example: Permutation Entropy

Larger permutation entropy means weaker predictability

- Permutation entropy = 1.00

- Permutation entropy = 1.75

- Permutation entropy = 2.29
Task: encode all ordinal patterns in a time series following an encoding scheme

Encoding scheme: map each ordinal pattern to a digitalised form

Example: permutation representation

123  132  213  231  312  321
Ordinal Pattern Encoding

- Problem: low computational efficiency for long time series
- Potential solution: pipelined accelerator
- Challenge: pipeline-friendly encoding scheme
  - Minimised data transmission
  - Simple data operations
  - Free from post-processing
Permutation representation is **NOT** pipeline-friendly

- Minimised data transmission?  
  No – too many bits needed
- Simple data operations?  
  No – requires sorting
- Free from post-processing?  
  No – requires complex hashing
Pipeline-Friendly Encoding Scheme

Proposed method: two-level-encoding scheme

- **Level 1:** Lehmer code for efficient computation
  \[
  \Rightarrow (1, 0, 0)
  \]

- **Level 2:** Compressed code for compactness and avoidance of post-processing
  \[
  (1, 0, 0) \Rightarrow 2
  \]
Level 1 Encoding: Lehmer Code

Given a sequence \( x = (x_1, \ldots, x_n) \) be a sequence of length \( n \), the Lehmer code of \( x \) a sequence in the form \( \mathcal{L}(x) = (l_1, \ldots, l_n) \) where

\[
l_i = \# \{ x_j : i < j, x_j < x_i \} \quad (1)
\]

Example:

- Sequence: \( x = (3, 8, 1) \)
- Ordinal pattern:

![Ordinal Pattern](image)

- Lehmer code: \( \mathcal{L}(x) = (1, 1, 0) \)
Key insight:
- Suppose that we have computed the Lehmer code for a subsequence
- It is possible to compute the Lehmer code for the next subsequence without sorting
Lehmer code representation is **NOT** pipeline-friendly

- Minimised data transmission?
  - No – too many bits needed
  - (fewer bits than permutations)
- Simple data operations?
  - Yes.
- Free from post-processing?
  - No – requires hashing
Level 2 Encoding: Compressed Code

- A Lehmer code uniquely maps to a number in the factorial number system
- Example

  \[(0, 1, 0) \implies 0 \times 2! + 1 \times 1! + 0 \times 0! = 1\]

- Key insight: it is possible to compress Lehmer codes using the factorial number system
Two-Level Encoding Scheme

Two-level encoding is pipeline-friendly

- Minimised data transmission?
  Yes – theoretical lower bound

- Simple data operations?
  Yes – inherited from Lehmer code

- Free from post-processing?
  Yes – directly applicable for histogram construction
Ordinal Pattern Encoding Engine

Data

Increment decider

Lehmer code reviser (adders)

0

Lehmer code reviser (buffers)

Compressor

Code
Experimental Evaluation

- **Hardware platform**
  - Maxeler MAX3 acceleration card with a Xilinx Virtex V6-SX475T FPGA
  - Intel i7-870 CPU cores with four physical cores (eight logical cores)

- **Data sets**
  - Financial data, EEG data, artificial data
  - Query orders: $n = 6, 12$
  - Lengths of time series: $T = 10^5, 10^6, 10^7, 10^8, 10^9$
Experimental Evaluation

Performance results

(a) Time, n=6
(b) Time, n=12
(c) SU, n=6
(d) SU, n=12

- Up to 62 times faster than one CPU thread
- Up to 12 times faster than eight CPU threads
- Up to 4 times more energy-efficient than eight CPU threads
Current and Future Work

- Further optimisation for the proposed architecture
- Application of the proposed architecture to other time series processing problems
- Self-adaptive decision support systems for time series prediction
Two-level ordinal pattern encoding scheme with pipeline-friendly properties
- Minimised data transmission
- Simple data operations
- Free from post-processing

Pipelined accelerator for ordinal pattern encoding

Experimental evaluation:
- Up to 12 times faster than four-core CPU
- Up to 4 times more energy-efficient than four-core CPU